What Does It Mean to be I/O Bound?

Finding Multi-threading Opportunities in Embedded Applications Using Intel® Software Products

White Paper

April 2008

Document Number: 319341-001

Contents

Figures

Revision History

Document Number	Revision Number	Description	Revision Date
319341	001	Initial release.	April 2008

1.0 Introduction

Software architects and programmers can no longer rely only on increases in processor clock frequency to provide gains in application performance. Mainstream computer system architectures, including many Intel product lines, have instead chosen a design philosophy of doing more work per clock cycle - at all levels of the platform. One consequence is that multiple processor environments have proliferated, and are now widely available for all markets, from server and mobile to embedded and communications. Thus, in order for a given application to take full advantage of these new systems, software should be split among multiple execution cores.

There are a couple of options to accomplish this partitioning, each with varying degrees of effort and performance payoff.

Many modern operating systems (OSs) support multi-processor environments natively; these are known as Symmetric Multi-Processing (SMP) OSs. If you are already using an SMP OS (or are planning to), it might be prudent to reap the benefits of allowing such an OS to schedule your applications across processor resources automatically, saving in-house programmers from needing to get into the details of software multi-threading. Another approach consolidates two or more OS's onto the same machine by virtualizing the underlying hardware with another layer of software (e.g., a VMM); here, each OS would run on their own core or cores.

However, these solutions might not meet your system requirements with respect to performance. For certain system workloads, multi-threading individual applications will be the best way to meet these goals. While this is the most difficult route, system architects and programmers will have the finest grain of control over how and where each application – more precisely, each thread of each application – will run on the platform.

This white paper demonstrates one methodology for analyzing a popular, open source, distributed storage application called Open-iSCSI for multi-threading opportunities. There are usually two overarching goals in this type of effort. The first is to find a way to reduce the amount of time it takes to complete a given set of work in the application (e.g., shorten data transfer times), and the second is to be able to run "more" in the same amount of time (e.g., increase data bandwidth). But these are not trivial tasks, especially for embedded systems that may use proprietary operating systems or have real-time requirements.

Realizing that this effort is challenging, Intel has created several industry-leading development tools to assist software architects and programmers as they multi-thread their applications. This family includes the Intel® C/C++ Compiler, the Intel® VTune™ Performance Analyzer and the Intel® Threading Tools (plug-ins for VTune™). This richly-featured environment is indispensable for writing and debugging the highest performing multithreaded code possible.

This document aims to show that these tools will help you to figure out where and how to thread an application…or if you even need to thread at all!

1.1 Why Open-iSCSI?

The proliferation of broadband networks has enabled video streaming become the hot application that it is. For many deployments of this type, the storage subsystem makes up much of the total system cost. With this in mind, iSCSI – a SAN protocol that deliver SCSI commands over TCP/IP networks – is a popular choice for such a system, given that it does not need its own dedicated infrastructure. In short, it is one of the most cost-effective and flexible storage options out there for a video streaming application.

Open-iSCSI is the most popular open source implementation of iSCSI and is therefore attractive for the usual list of open source benefits (e.g., royalty-free licensing, updates by an open community of programmers, etc.). The application itself is made up of two main portions, an "initiator" (which serves the same purpose as an SCSI bus adaptor) and a "target" (which is the storage resource). The initiator is further partitioned into user and kernel parts, where the kernel portion implements the bulk of the I/O path (i.e., the iSCSI Read and iSCSI Write) and the user portion contains the management and control plane functions.

This paper focuses on our analysis work with respect to the Open-iSCSI initiator executables. Typically, applications that are CPU-bound – those that have working sets that mostly fit within the CPU cache - make the best candidates for multi-threaded implementations. However, we would also like to determine if the Open-iSCSI initiator, which is known to have a high level of I/O activity (to and from the network), can also lend itself to a multi-threaded implementation on a multi-core platform.

1.2 Target Audience

This white paper targets software developers wishing to optimize existing applications for performance on Intel® Multi-Core systems. More specifically, those looking to write multi-threaded software for embedded systems will find particular value in the methods described below.

The paper is subdivided into the following sections:

- Section 2: Determining Baseline Performance
- Section 3: Performance Analysis Using Intel® Software Tools
- Section 4: Analyzing Application Structure
- Section 5: Assessing Options for Multithreading

Throughout this paper, we use Intel® Software Tools – in particular, the Intel® VTune™ Performance Analyzer – to find so-called hotspots in the Open-iSCSI initiator application; these hotspots are areas of code that warrant further study because they are perceived as a bottleneck for one reason or another (i.e., inefficient use of system resources).

2.0 Determining Baseline Performance

Each application has a set of criteria by which its effective performance is measured (e.g., how quickly it can perform a particular function, or how much data it can process in a given set of time); one of the first steps for analysis is to benchmark the original application (i.e., existing and unmodified) using these criteria.

With respect to the Open-iSCSI initiator, our focus will be on read throughput of the Initiator host (higher numbers are better).

This section describes the development environment – both hardware and software – and provides instructions to produce and measure these performance metrics.

2.1 Development Environment Setup - Hardware

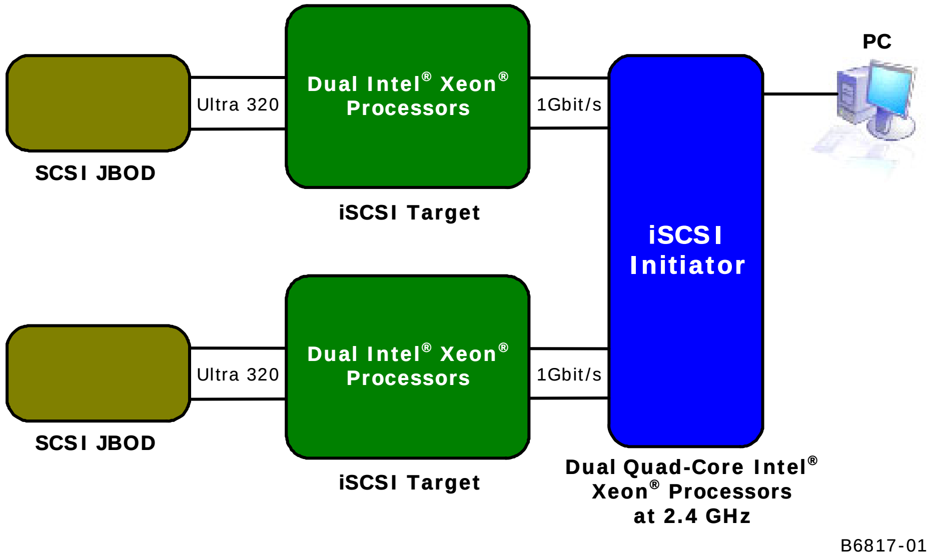
The following hardware elements were used for the analysis:

- One reference board based on Dual Quad-Core Intel® Xeon® processors at 2.4 GHz
- Two servers based on Dual Intel® Xeon® processors at 3.2 GHz
- Two Dell PowerVault* 220S Disk Storage Enclosures, each with 14 SCSI Ultra 320 drives
- One Microsoft Windows* based desktop computer with an Intel® Core™ 2 Duo processor
- One Extreme Networks 400-48t gigabit Ethernet switch

The Bensley reference board is used as the Open-iSCSI initiator and the two Langley servers are used as its iSCSI targets. Each iSCSI target server is connected to one of the Dell PowerVault 220X JBOD storage enclosures (and, by association, a set of Ultra 320 SCSI hard drives). The system also includes a Windows*-based desktop computer, used to remotely run and receive VTune™ analysis data from a Linux-based VTune™ data collector running on the iSCSI Initiator.

The full system block diagram is as follows:

Figure 1 Open-iSCSI initiator development environment



2.2 Development Environment Setup – Software

The following software elements were used for the analysis:

- RedHat AS4, Update 4 with kernel 2.6.16-21-0.8
- SUSE EL 10 with SP1
- Microsoft Windows XP* Professional Edition with SP2
- Open-iSCSI version 2.0-865.9
- Intel® VTune™ Performance Analyzer version 9.0
- IOTest 3.1 (benchmarking for SCSI devices)
- iscsitarget-0.4.15 (driver to export iSCSI devices to network)
- mdadm-2.6.3 (soft RAID management)

Install this software to the hardware components as indicated Table 1:

Hardware System		Necessary Software
Reference board based on Dual Quad-Core Intel® Xeon® processor at 2.4GHz		SUSE EL10 with SP1 Open-iSCSI version 2.0-865.9 IOTest 3.1 mdadm-2.6.3 Intel® VTune™ Performance Analyzer version 9.0 *remote data collector for Linux*

Server based on Dual Intel® Xeon® processor at 3.2GHz		RedHat AS4, Update 4 with kernel 2.6.16-21-0.8 iscsitarget-0.4.15 mdadm-2.6.3
Microsoft Windows*-based Desktop computer with an Intel® Core™ 2 processor		Windows XP Professional Edition with SP2 Intel® VTune™ Performance Analyzer version 9.0

Note: Version 2.0-865.9 of Open-iSCSI was the latest version available at the time of writing.

2.3 Benchmark Procedure

We are most interested in the read throughput of the iSCSI initiator. The following procedure to sets up the system to collect this data:

On the iSCSI target machine:

Create two Soft-RAID disks on each Langley server (Level 0, chunk size 256KB). In our hardware configuration, each Soft-RAID disk includes seven hard drives.

```
[root@rms021 ~]# cat /proc/mdstat
Personalities : [raid0]
md1 : active raid0 sdj[0] sdp[6] sdo[5] sdn[4] sdm[3] sdl[2] sdk[1]
      248964352 blocks 256k chunks

md0 : active raid0 sdc[0] sdi[6] sdh[5] sdg[4] sdf[3] sde[2] sdd[1]
      248964352 blocks 256k chunks
```

Start the iSCSI target service to expose these two Soft-RAID disks to the network.

```
Target iqn.2007-08.com.intel:rms021.target01.disk1
        Lun 0 Path=/dev/md0,Type=blockio
Target iqn.2007-08.com.intel:rms021.target01.disk2
        Lun 0 Path=/dev/md1,Type=blockio
```

On the iSCSI initiator machine:

1. Bind the iSCSI interface with a selected NIC. For instance, for eth2 /etc//etc/iscsi/ifaces/iface.2 was created.
    a. Create an iSCSI binding file under the /etc/iscsi/ifaces/ directory with the following contents:

```
# iface.transport_name = tcp

# This values is required and valid values for iface.transport_name are:
# - tcp (Software iSCSI over TCP/IP)
# - iser (Software iSCSI over infinniband
# - qla4xxx (Qlogic QLA4XXX HBAs)

# __One__ of the following values are required for the binding.
#
# To bind by network interface name (example: eth0, eth2:2, eth1.3)
# set iface.net_ifacename
# example:
 iface.net_ifacename = eth2

# To bind by hardware address set the NIC's MAC address to iface.hwaddress
# example:
 iface.hwaddress = 00:15:17:1D:84:84
```

In this example, the iSCSI device is bound to the “eth2” network interface.

2. Start the iSCSI initiator service.

   a. Type “/etc/init.d/open-iscsi restart” at any command prompt.

```
root      5717     1  0 12:58 ?        00:00:00 /sbin/iscsid -c /etc/iscsi/iscsid.conf -p /var/run/iscsi.pid
root      5718     1  0 12:58 ?        00:00:00 /sbin/iscsid -c /etc/iscsi/iscsid.conf -p /var/run/iscsi.pid
```

3. Login into both iSCSI targets (each network interface serves one iSCSI disk array) and create one Soft-RAID (Level 0, chunk size 1792KB) with four disks in total; two of the four disks will be from one iSCSI target and two from the other.

   a. Go to “/root/scripts” and type “./iscsiadm.sh” at the command prompt.

```
Bensley-Initiator:~/scripts # sh iscsiadm.sh
172.19.1.2:3260,1 iqn.2007-08.com.intel:rms025.target02.disk1
172.19.1.2:3260,1 iqn.2007-08.com.intel:rms025.target02.disk2
172.19.2.2:3260,1 iqn.2007-08.com.intel:rms025.target02.disk1
172.19.2.2:3260,1 iqn.2007-08.com.intel:rms025.target02.disk2
172.20.1.2:3260,1 iqn.2007-08.com.intel:rms021.target01.disk1
172.20.1.2:3260,1 iqn.2007-08.com.intel:rms021.target01.disk2
172.20.2.2:3260,1 iqn.2007-08.com.intel:rms021.target01.disk1
172.20.2.2:3260,1 iqn.2007-08.com.intel:rms021.target01.disk2
Login session [iface: iface.4, target: iqn.2007-08.com.intel:rms021.target01.disk1, portal: 172.20.1.2,3260]
Login session [iface: iface.5, target: iqn.2007-08.com.intel:rms021.target01.disk2, portal: 172.20.2.2,3260]
Login session [iface: iface.2, target: iqn.2007-08.com.intel:rms025.target02.disk1, portal: 172.19.1.2,3260]
Login session [iface: iface.3, target: iqn.2007-08.com.intel:rms025.target02.disk2, portal: 172.19.2.2,3260]
Bensley-Initiator:~/scripts #
```

The screen capture above shows “successful” output from the iscisadm.sh script.

   b. Return to home directory and type “mdadm -Cv -n 4 -l 0 -c 1792 /dev/md0 /dev/sd[b-e]” at the command prompt.

   c. Verify the RAID state by typing “cat /proc/mdstat” at any command prompt.

```
Bensley-Initiator:~/scripts # cat /proc/mdstat
Personalities : [raid0]
md0 : active raid0 sdc[0] sdf[3] sde[2] sdd[1]
      995852288 blocks 1792k chunks

unused devices: <none>
Bensley-Initiator:~/scripts #
```

4. Start I/O read transactions via IOTest.

   a. Go to the “tools/iotest/iotest31/” directory and type “./IOtest” at the command prompt.

5. Collect I/O throughput and CPU utilization with iostat and mpstat, respectively.

   a. Type “./iostat” and “./mpstat” at any command prompt, respectively.

2.4 Initial Benchmark Results

Figure 2 shows the I/O Read throughput results and Figure 2 shows the associated CPU Utilization rate.

Figure 2 I/O Read throughput

```
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          18.50    0.00  202.50   16.50    0.00  562.00

Device:    rrqm/s wrqm/s   r/s   w/s  rsec/s  wsec/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await  svctm  %util
md0          0.00   0.00 89600.00  0.00 716800.00    0.00 358400.00     0.00     8.00     0.00    0.00   0.00   0.00
```

Figure 3 CPU Utilization

```
11:25:54     CPU   %user   %nice    %sys %iowait    %irq   %soft  %steal   %idle    intr/s
11:25:56     all    2.37    0.00    9.24    2.00    0.37   17.43    0.00   68.58  15462.00
11:25:56       0    0.00    0.00    0.00    0.00    0.00    0.00    0.00  100.00     14.50
11:25:56       1    3.50    0.00    1.00    0.00    1.00   43.00    0.00   51.50   3372.50
11:25:56       2    0.00    0.00   13.50    0.00    0.00    0.00    0.00   87.00      8.00
11:25:56       3    3.00    0.00    2.50    0.00    0.50   30.00    0.00   63.50   3486.00
11:25:56       4    5.50    0.00   10.00    0.00    0.00    0.00    0.00   84.50      0.50
11:25:56       5    0.00    0.00    0.00    0.00    0.00    0.00    0.00  100.00    251.00
11:25:56       6    5.00    0.00    2.00    0.00    1.00   30.50    0.00   62.00   4325.00
11:25:56       7    1.50    0.00   46.00   16.00    0.50   36.50    0.00    0.00   4003.00
```

For our setup, iostat yielded:

- iSCSI Read rate: 358.0 MB/s

And mpstate yielded:

- Overall CPU utilization rate: 31.70%

From this point forward, the iSCSI read rate and the CPU utilization are, basically, the “performance metrics to beat”. And, as such, we will keep them in mind as we start our analysis with the Intel® Software Tools. Please note that while the analysis techniques discussed here are applicable to a board range of needs, you may be

interested in capturing other performance metrics, depending on your system requirements, however they will be beyond the scope of this paper. Please see the Intel® VTune™ Performance Analyzer documentation for more information on the full range of its capabilities.

3.0 Performance Analysis Using Intel® Software Tools

Intel® VTune™ Performance Analyzer is a suite of tools that allow one to easily collect, display, and organize run-time behavioral information on a given application, mostly for performance tuning purposes. And while these tools are comprehensive and highly-configurable, the most common and useful functions are accessible through a set of “wizards”. Intel recommends that you start your application analysis by running one or more of these wizards, which automate the collection and presentation of these parameters. If more complicated or detailed information is needed, each of the tools within Intel® VTune™ Performance Analyzer environment is easily configurable.

Following this advice, we first employed the “Sampling Wizard” to conduct a remote system analysis of the Open-iSCSI initiator running on the Intel® Xeon® Processor 5100 series reference board, and later, we used the “Call Graph Wizard” to understand more about the software architecture of the application.

Note: Any executable can be analyzed to at least some degree via the Intel® VTune™ tools, regardless of its origin. But to collect more useful information, the executable should be built with a “DEBUG” configuration (i.e., minimal optimizations, includes source, etc.) and, further, with the “/fixed:no” linker option; this option is required for the level of analysis described in this paper.

We ran all the tools with their default data collectors and environmental variables intact. For more detailed information on starting and running the wizards, please reference the Intel® VTune™ Performance Analyzer documentation.

3.1 Procedure for Analysis – Sampling Wizard

The following instructions for the iSCSI target are exactly the same as found in the Benchmarking section; and the instructions for the iSCSI initiator are basically the same, but with the addition of starting the VTune™ collection daemon. In addition, there are extra steps for the Windows* machine in order to download and display the Sampling data.

On the iSCSI target machine:

1. Perform the exact same setup as described in the Benchmark section.

On the iSCSI initiator machine:

1. Start the iSCSI initiator service.
    a. Type “/etc/init.d/open-iscsi restart” at any command prompt.
2. Log in into both iSCSI targets and create one Soft-RAID (Level 0, chunk size 1792KB) with four disks in total; two of the four disks will be from one iSCSI target and two from the other.

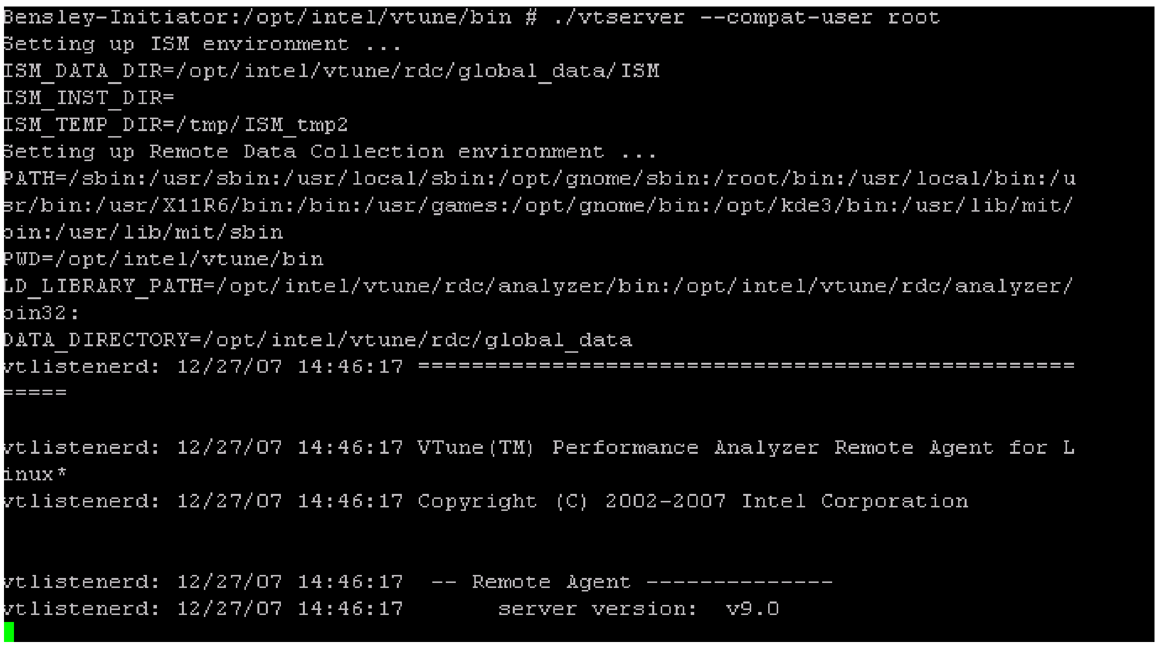
a. Go to “/root/scripts” and type “./iscsiadm.sh” at the command prompt.

b. Return to home directory and type “mdadm -Cv -n 4 -l 0 -c 1792 /dev/md0 /dev/sd[b-e]” at the command prompt.

c. Verify the RAID state by typing “cat /proc/mdstat” at any command prompt.

3. Start the VTune™ Remote Agent service.

a. Go to the “/opt/intel/vtune/bin/” directory and type “./vtserver --compat-user root -d /path/to/data/directory” at the command prompt, where “/path/to/data/directory” is where you want VTune to store its (potentially large) analysis data.



```
Bensley-Initiator:/opt/intel/vtune/bin # ./vtserver --compat-user root
Setting up ISM environment ...
ISM_DATA_DIR=/opt/intel/vtune/rdc/global_data/ISM
ISM_INST_DIR=
ISM_TEMP_DIR=/tmp/ISM_tmp2
Setting up Remote Data Collection environment ...
PATH=/sbin:/usr/sbin:/usr/local/sbin:/opt/gnome/sbin:/root/bin:/usr/local/bin:/u
sr/bin:/usr/X11R6/bin:/bin:/usr/games:/opt/gnome/bin:/opt/kde3/bin:/usr/lib/mit/
bin:/usr/lib/mit/sbin
PWD=/opt/intel/vtune/bin
LD_LIBRARY_PATH=/opt/intel/vtune/rdc/analyzer/bin:/opt/intel/vtune/rdc/analyzer/
bin32:
DATA_DIRECTORY=/opt/intel/vtune/rdc/global_data
vtlistenerd: 12/27/07 14:46:17 ==================================================
=====

vtlistenerd: 12/27/07 14:46:17 VTune(TM) Performance Analyzer Remote Agent for L
inux*
vtlistenerd: 12/27/07 14:46:17 Copyright (C) 2002-2007 Intel Corporation


vtlistenerd: 12/27/07 14:46:17  -- Remote Agent --------------
vtlistenerd: 12/27/07 14:46:17       server version:  v9.0
```

On the Windows machine:

1. Start VTune™; an “Easy Start” window should pop up. Click on the “New Project” button.

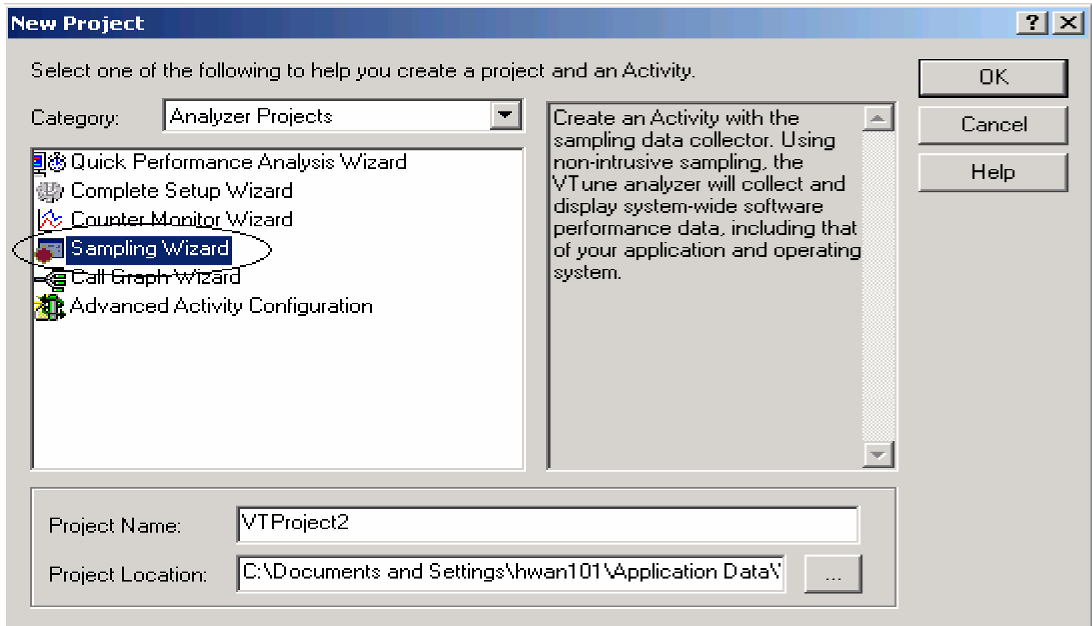
Note: If VTune™ is already open, simply click on File > “New Project…” from the top menu.

2. Create a new Sampling Wizard project.

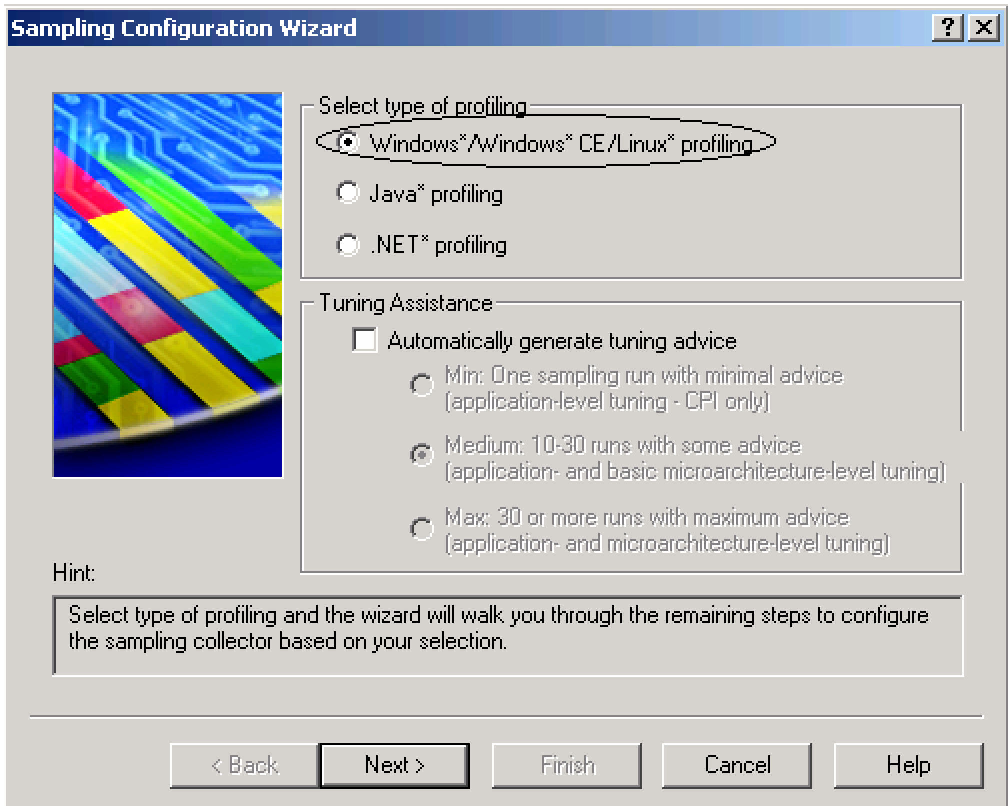
a. In the “New Project” window, be sure the “Analyzer Projects” is displayed in the “Category” drop-down menu, and choose “Sampling Wizard” from the list of project types.

b. Type in a new “Project Name” and choose a new “Project Location” directory, if desired. For this paper, we will use “open-iscsi” as the project name, and the default directory as the project directory.

c. Press the “OK” button.

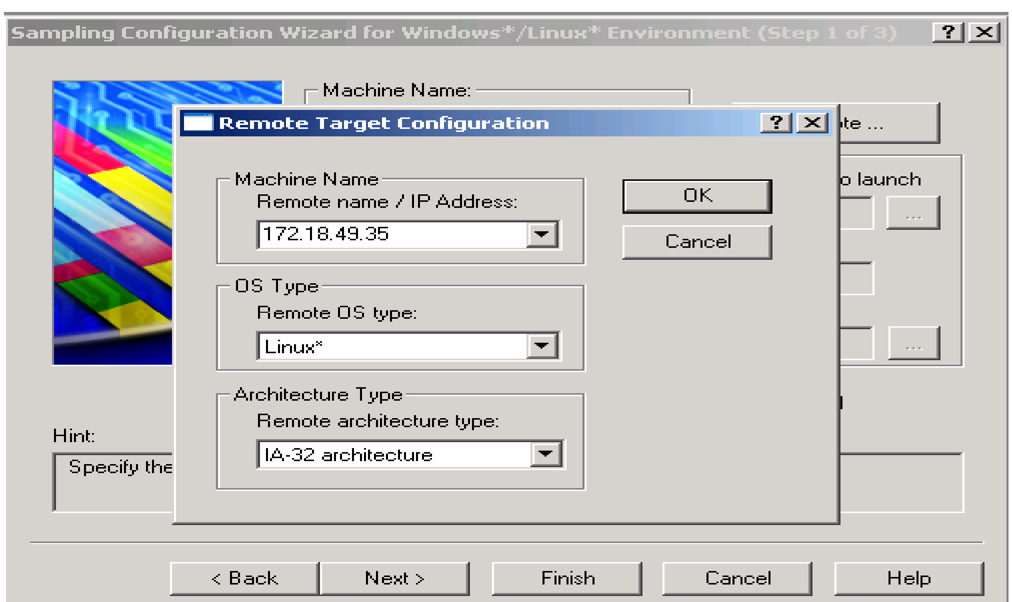


3. Leave the defaults on the Sampling Configuration Wizard window.
    a. “Windows*/Windows* CE/Linux* profiling”
    b. “Tuning Assistance” off (i.e., unchecked)
    c. Press the “Next” button.



4. Finish the Sampling Wizard configuration.

   a. Press the "Remote…" button and enter the IP address of the iscsi initiator machine; press "OK" to close this window.



   b. Click the "No application to launch" box.

c. Leave the rest of the defaults:

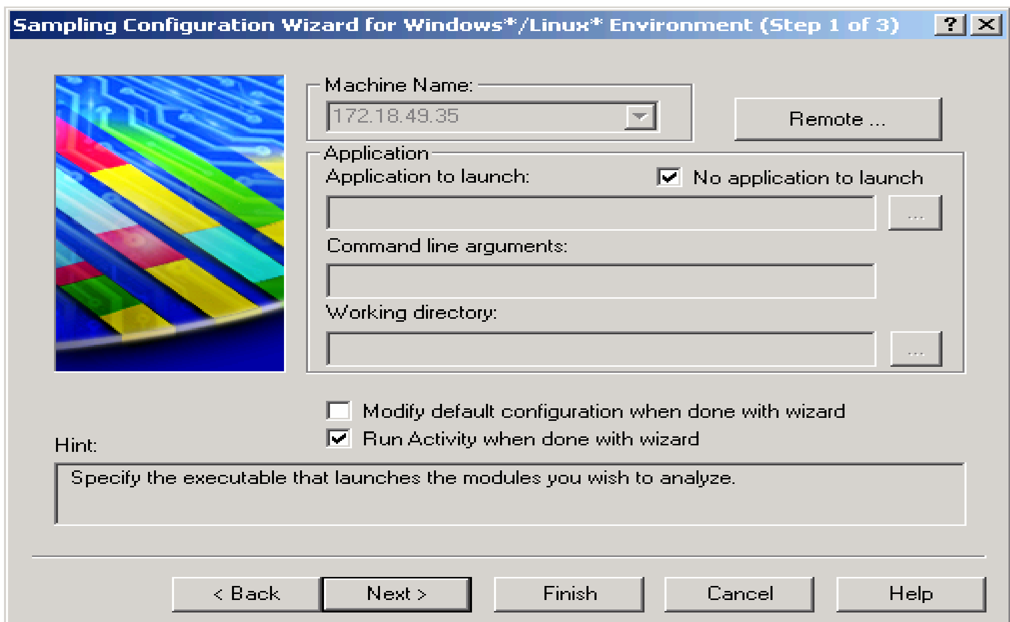
i. "Modify default configuration when done with wizard" should be unchecked.

ii. "Run Activity when done with wizard" should be checked.

5. Back on the iSCSI initiator machine, start the I/O read transactions via IOTest.

   a. Go to the "tools/iotest/iotest31/" directory and type "./IOtest" at the command prompt.

6. Return to the Windows* machine and Press the "Finish" button on the VTune™ wizard.



At this point, VTune™ will start a Sampling analysis using the configuration values we specified with the wizard. Since VTune™ will not be starting the Open-iSCSI initiator application itself ("No application to launch"), it immediately begins collecting data from the remote agent running on the iSCSI initiator machine. And the analysis will run until we terminate it explicitly.

7. After IOTest is complete, press the red "Stop Activity" button from the toolbar at the top of the VTune™ window. Alternatively, click on "Activity > Stop" from the top menu selections.

   a. You may wish to restart the IOTest to get more data before stopping the VTune™ analysis.

8. The first Sampling results will be automatically displayed in the main VTune™ window. In addition, each run of the tool (if you later run it again) will be saved and available for viewing later via the "Tuning Browser".

Note: VTune™ functions can be invoked via a command line version of the tool – as opposed to the GUI method described above. Please see the VTune™ documentation for more information.

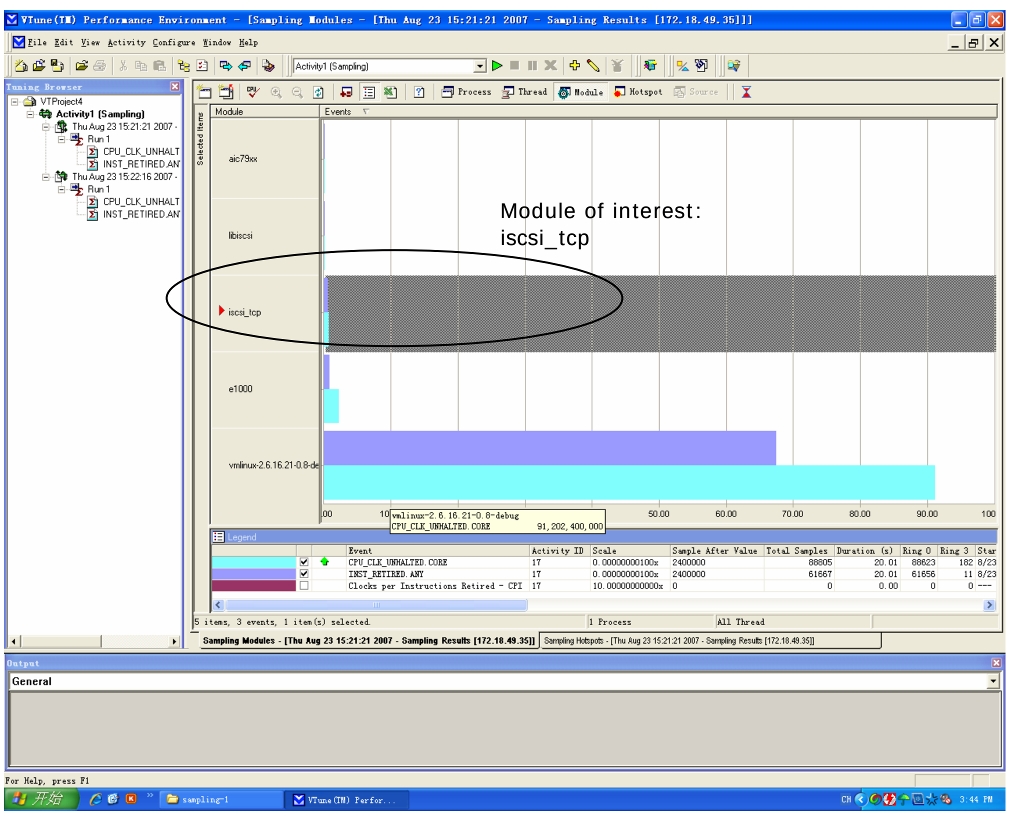
3.2 Results of Analysis – Sampling Wizard

As mentioned before, the default settings for the Sampling Wizard automatically includes a few system counters to be collected, such as “CPU cycles per instruction.” Once a Sampling run is complete, the Intel® VTune™ Performance Analyzer GUI automatically displays the results graphically in a histogram format. That is, the first view of the Sampling data shows us – on a “Module” by “Module” basis – which executable or library recorded the highest levels of the specified metrics. In this way, one can, for example, pay attention to the “CPU cycles per instruction” data and easily figure out where the processor is spending most of its time during the test run of the application.

Note that the tool will capture – and display – everything that was running in the system at the time of the test, which would include the Intel® VTune™ Performance Analyzer itself as well as various Operating System libraries.

Figure 2 shows the top level Sampling results view (i.e., “Module” view); and except for the Intel® VTune™ data collector and e1000 driver operation, the iscsi_tcp executable of the Open-iSCSI initiator takes the most CPU cycles.

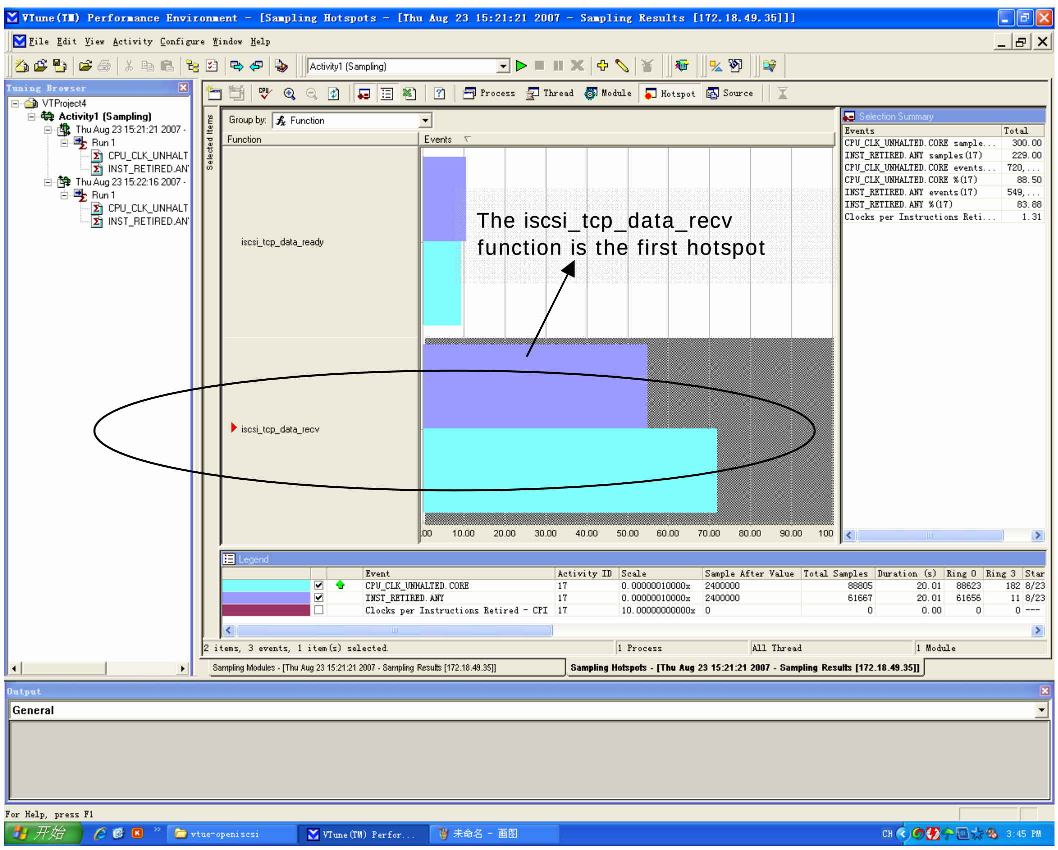
Figure 4 Sampling Results for open-iscsi initiator in Module View



Double clicking on the module of interest – in this case, the iscsi_tcp module, which is highlighted in Figure 4 – the display "drills down" into a Function-by-Function view of the Sampling data. An alternative method to drill down is to highlight the module of interest (i.e., single-click on it), and then click on the "Hotspot" icon on the top menu bar of the Sampling Results sub-window.

Figure 3 shows us this view for the iscsi_tcp module. The sampling results in this view show that the number one "hotspot" from a CPU count point of view is in the function "iscsi_tcp_data_recv"; this function will be our "area of interest" for performance tuning.

Figure 5 Sampling Results for open-iscsi initiator in Function View



This information is useful for the developer insofar as it provides empirical data relevant for making decisions about where to spend engineering resources. In general, you will want to focus your time on the areas of the application with high sample counts, whether these are CPU cycles, bus bandwidth measurements or any of the other metrics available via the VTune™ Sampling tool.

3.3 Summary of Analysis – Sampling Wizard

By focusing on the "CPU cycles per instruction" data counter, the Sampling Wizard of the Intel® VTune™ Performance Analyzer has provided enough data for an initial attempt at multi-threading the Open-iSCSI initiator application. Specifically, the

Sampling tool identified the function where the processor spends most of its time during our test run: "iscsi_tcp_data_recv".

Note: We also mentioned that there are other metrics that can be collected via the Sampling tool, including effective cache usage and bus access frequency. These aspects of system behavior also warrant further study if, after looking at areas of high CPU utilization, more performance is desired (or there was nothing to be done with respect to high CPU utilization). However, such discussion is beyond the scope of this paper.

The next step in the optimization process is to determine how this function relates to the structure of the application at large. Ultimately, we would like to take the sections of code with the highest CPU cycle counts and partition them across our multiple cores so that they may execute in parallel.

The Intel® VTune™ Performance Analyzer provides a tool that can help answer this question in certain situations – the "Call Graph". The Call Graph tracks the function call tree for a given executable at run-time and maps these relationships graphically in the VTune™ GUI for the developer. In other words, if the Sampling tool can help us to find where to focus our attention with respect to optimizing application performance, the Call Graph tool can help us to figure out what to do about it.

However, because there is no specific "executable" that we can specify to VTune™ - the Open-iSCSI initiator code is in the kernel – we will not use the Call Graph tool. We have the design documents for the application and will use them to figure out if and how we can multithread around the iscsi_tcp_data_recv function.

4.0 Analyzing Application Structure

It is possible to get better application performance by splitting up code with high CPU_CLK counts and distributing its work among multiple cores. That can be easier said than done.

First, we'll need to answer the following questions:

1. What are the computational or I/O-based dependencies within the function?
2. Will the overhead of threading overshadow its benefits?

For many applications, the Call Graph tool within VTune™ can help us answer these questions. However, given that the Open-iSCSI initiator application is a kernel module, we will not be able to use it for our study today; as implied, the Call Graph tool requires a specific and stand-alone executable on which to perform its analysis.

Note: In contrast, the Sampling tool collects system-wide data (i.e., it does not need to be tied to a specific executable by default), which is why we were able to use it for the first part of our study.

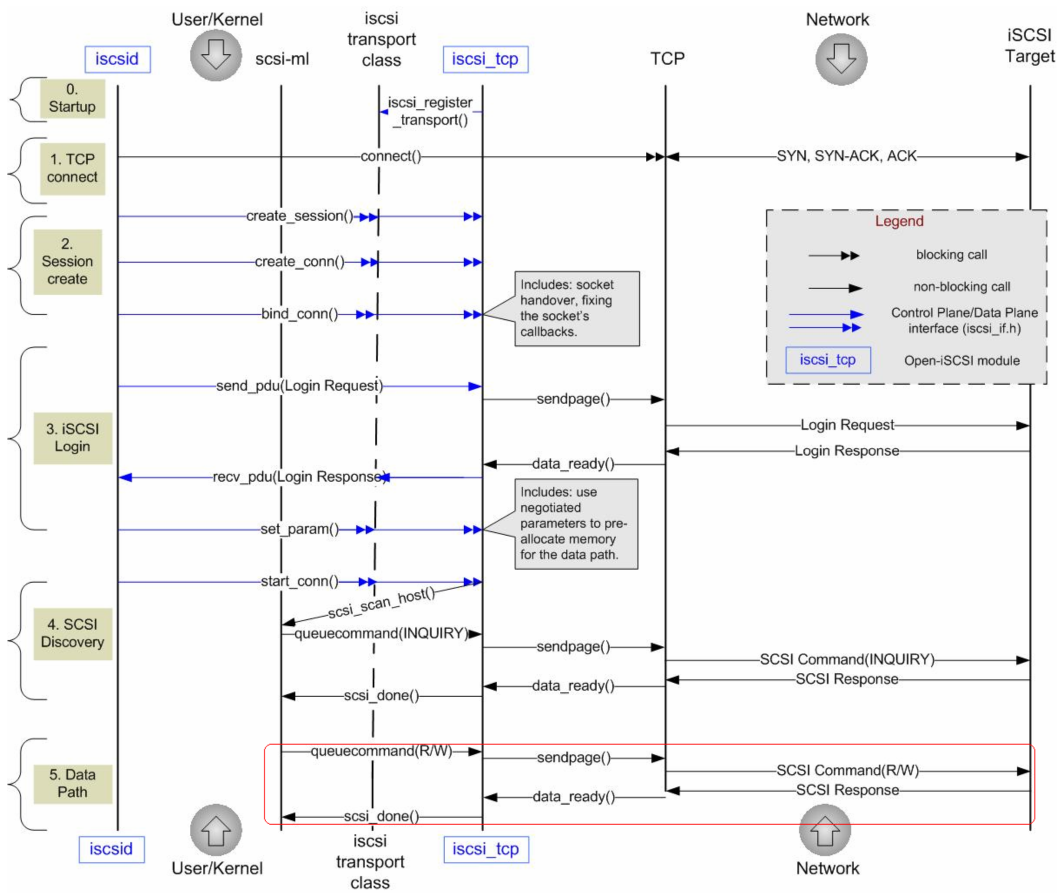
Instead, we will read the publicly-accessible design documents as well as the application source code in order to reason how best to multi-thread within or around the iscsi_tcp_data_recv function. The Call Graph tool would provide this level of information – the calling hierarchy of the application – but give it to us while also recording the frequency of each function access at run-time for a richer and "more real" version of events (given "real" application inputs).

That said, the Open-iSCSI initiator software architecture is such that the static design documents, along with the source code, made it apparent how to proceed.

4.1 Analysis Results – Application Structure

The Open-iSCSI initiator has a Request/Response-type architecture. Figure 6 shows the Open-iSCSI initiator function call logic, with a red box highlighting the data path as Step #5.

Figure 6 Open-iSCSI initiator function call logic



By looking at its source code, the “iscsi_tcp_data_recv” function – part of the iscsi_tcp module mentioned in Figure 4 above – calls another function named “skb_copy_bits”, which, in turn, spends much of its time waiting for data from the network. To relate this to the function call logic of Figure 5, “iscsi_tcp_data_recv” spends most of its time (via child functions) waiting for I/O transactions between the “SCSI Command (R/W)” and “SCSI Response” steps of the Data Path.

4.2 Summary of Analysis – Application Structure

The VTune™ Sampling tool helped us find the function that was taking up most of the computational cycles in the Open-iSCSI initiator application. And this function became our focus for the next step of the study: how to split up its work over multiple threads, presumably to be run on multiple processing cores.

Since we were not able to use the VTune™ Call Graph tool, we determined the role of “iscsi_tcp_data_recv” in the application via design documents and source code. Ultimately, we found that the application has I/O-based dependencies, meaning that optimizing the initiator code without increasing data access would not get us overall application gains. The application structure analysis showed us that, in fact, there were no substantial computational activities within the “iscsi_tcp_data_recv”

function. Rather, we found that the bulk of the time spent there is actually the result of another function, named “skb_copy_bits”, which performs memory copies from the network to the CPU memory space.

In other words, the hotspot that we identified with the Sampling tool (i.e., “iscsi_tcp_data_recv”) is not a bottleneck in the system, because it does nothing – it has nothing to do – while it waits for I/O transactions to take place. This implies that the Open-iSCSI initiator is so-called I/O-bound (rather than CPU-bound), making it not suitable for multithreading on a per-instance basis.

At this point, we should be careful to note that we are not at a dead-end with respect to improving the performance of this application even as we have come to the conclusion that we must consider some other alternatives to multithreading this part of the application.

5.0 Assessing the Options and Next Steps

This paper attempts to document multithreading of the Open-iSCSI Initiator application – or, rather, seeing if it was possible to multithread. We set up test systems, captured initial benchmarks, and analyzed system performance and structure using both the Intel® VTune™ Performance Analyzer and more traditional code inspection methods. Ultimately, we determined that this application is not a good candidate for multithreading. But that does not mean that these procedures were a wasted effort.

The methodology documented here can be applied to any application analysis for performance. In this particular case, the result was that the Open-iSCSI initiator application was I/O bound, and therefore not worth spending any more time trying to multithread – there would be no performance gain without optimizing the I/O facilities of the system first. This actually implies that there is headroom to run other CPU-intensive applications, like content processing, on the platform in addition to Open-iSCSI.

Given the situation, one approach is to take a step back from looking at these individual applications to examine a fuller view of platform resources, including CPU cycles, memory bandwidth and various I/O interfaces. For example, perhaps the maximum transfer speed measured via IOTest during the Benchmarking stage is acceptable on a per-instance basis of the Open-iSCSI application. If that is the case, running multiple copies of it, each servicing a different network interface, would it meet product requirements?

Or, given that there are plenty of CPU cycles to handle the existing packet bandwidth, another possibility is to add new data processing code to the Open-iSCSI application itself. Or, start new applications on the platform (ones that don't access the same I/O interfaces, anyway). No matter the approach, your software will be doing more with the same hardware.

A full discussion of potential solutions is beyond the scope of this paper, but note now that the Intel® VTune™ Performance Analyzer can be applied to a full range of software and hardware architectures. The work documented in this paper has just grazed the surface of the rich feature set of the Intel® Software Tools.

6.0 Related Links

Intel® Software Products http://www.intel.com/software/products

Intel® Software Network http://softwarecommunity.intel.com/isn/home/

Intel Press http://www.intel.com/intelpress/sum_swcb2.htm